ViSS Technologies

ViSSAPI Specification and User Manual

API (Application Programming Interface) Specification and User Manual for innovative functionality on Video Indexing and Search, provided by ViSS (Video Search Systems)

www.viss-tech.com

Revision 1.30

November 14, 2012

1. Document Specification

Document Attribute	Document Attribute Value	
Title	ViSSAPI Specification and User Manual	
CSCI[1]	02-00-00-0A2F-01-30	
Category	User Manual, Technical Specification	
File	ViSSAPIUserManual_v_1_2_withViSSExtSearch_DO_NOT_DELETE.docx	
Pages	24	
Distribution		
Author	ViSS Technical Department	
Distribution Characteristics	Public	
Distribution List	Public	
Status		
Approvals		**Date**
Project Manager	Federico Carrasco (CTO)	Nov. 14, 2012
Project Quality Controller	Federico Carrasco (CTO)	Nov. 14, 2012

[1] Computer Software Configuration Item

2. Table of Contents

3. Executive Summary

This document describes the ViSSAPI (Application Programming Interface). This ViSSAPI provides the required set of functions for video indexing and video search.

With this API, ViSS introduces to the video industry and community a new, ultra-fast, patent applied, stream based, multimodal video search engine, with an innovative method for characterizing (fingerprinting) video segments.

This API enables fast creation of fingerprint database (indexing functionality) and fast video retrieval in a video repository (search functionality).

The ViSSAPI is a set of C/C++ functions, easy to integrate in C/C++ custom applications.

This document also describes the steps required by the API user, in order to create custom C/C++ applications that handle (index and search) video files.

The user of this API should be familiar with C/C++ programming and relational databases. All examples and coding has been tested and evaluated using Microsoft Visual Studio 2010 and SQL Server Management Studio 2008 (SQL Server Code Name “Denali” CTP3).

4. Contact information and support

ViSSAPI has been designed and developed by ViSS (Video Search Systems), an early stage video processing company with over two years of innovative research in the development of a revolutionary core technology, acting as a versatile multimodal video search engine. Our technology provides unique solution to the Media Content retrieval problem.

We are Silicon Valley based company located at Sunnyvale CA, (440 N. Wolfe Rd. - Sunnyvale, CA 94085) constantly evolving our core technology to meet the demanding field of video and media content.

Our team of engineers will provided you top level support related on our API, with samples, and solutions to your technical enquires.

www.viss-tech.com
ViSS (Video Search Systems)
440 N. Wolfe Rd. Sunnyvale, CA 94085
federico@viss-tech.com | +1 408 459 1504

5. Architecture

This section describes in brief the ViSSAPI technology and architecture.

5.1 ViSSAPI Innovation

ViSSAPI offers a comprehensive solution for Direct Video Search (DVS), based on patent-pending technology for characterizing video streams. It searches video content through a stream alignment process without use of any kind of metadata information.

ViSSAPI uses event occurrences associated with an expandable set of event attributes. In its basic form it matches durations between events under predefined loss and miss-insertion tolerances. The concept can be tightly coupled with scene changes as detected in video.

ViSSAPI can be used by engineers and designers through a dynamic link library (DLL) and companion MSSql database.

Events and event attributes are independently and automatically extracted from common video stream formats and then inserted into the database.

Video search starts from a query clip and occurs exclusively via automatically compiled queries. It returns the video film and position where the query clip occurs.

ViSS offers in addition to ViSSAPI a comprehensive design, development, test and deployment .NET environment for integrating its components with other third party developments sharing its approach on stream matching and alignment of sequences.

ViSSAPI can also serve as the underlying platform for elaborate content annotation, feature based search and video material editing, as well as for regrouping of video material according to higher level criteria. Independent text, sound, image and video annotation and search mechanisms can be easily integrated with the internal indexing as employed in ViSSAPI.

Based on the next figure, the main parts of the video indexing and video searching architecture are elaborated:

Fingerprint Extracting & Video Indexing

This is the process by which any video file is entered and populates the database.

For entering the ViSSAPI environment the video file in any format is serially processed, whereby events and event attributes are extracted. These constitute the fingerprint, are mapped into the video index and then inserted into the database. Parameters for the fingerprint extracting and indexing process are of no concern to the user. Upon completion of this process the original video file is of no further use.

All videos, whether representing content or the search video clip(s), undergo the same fingerprint extracting and indexing process and are stored in an unified way into the database.

Searching

Any video in the database, or part thereof, can be used as a query. With default settings the user simply identifies the video constituting the query. If desired, only a segment of this video is considered as a query. Moreover event attributes used in the query as well as various tolerance and result reporting parameters can be user defined.

Typical event attributes are scene durations, colors, movement activity, either individually or in any combination.

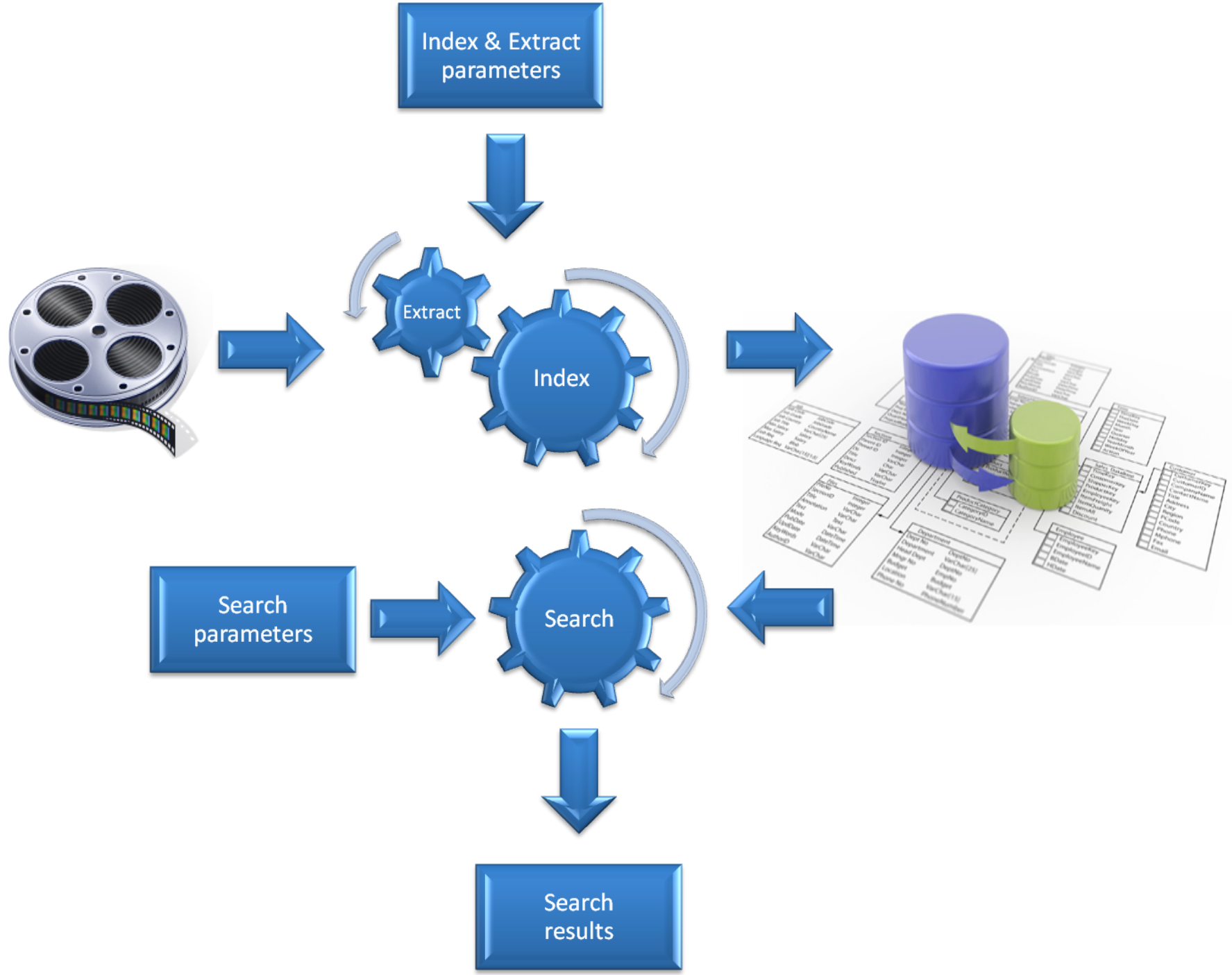
Key features of the ViSSAPI technology is the embedded resilience to mismatches. A query clip being a replica of some part of the database might have been received in another format, different quality, bearing different logos/subtitles. However stream alignment as used for search takes into account event losses and miss-insertions. Event attributes have their own tolerances, while movement activity can, for example, substitute for the irrelevance of color information. Thus a black and white query clip can be found within its colored full length origin by simply not using color information.

By virtue of the inherent event loss and miss-insertion resilience, scene durations alone are adequate to give accurate results, provided that the query clip is of nontrivial duration (5 or more scene changes).

Searching one-step:

A one-step procedure, if preferred, can be chosen by the user. He simply submits the video file constituting the query and obtains directly the results. All above steps are followed transparently without the submitted query clip to remain in the database.

Search result(s) give the video(s) and position(s) therein, where the query clip has been found. Typically an advertisement clip is searched within a database repository consisting of thousands of hours of stored TV program(s). According to the naming conventions and durations of individual videos used in the database, such a query will be located in many videos and in every such video, in various positions. A position is represented as the frame number.

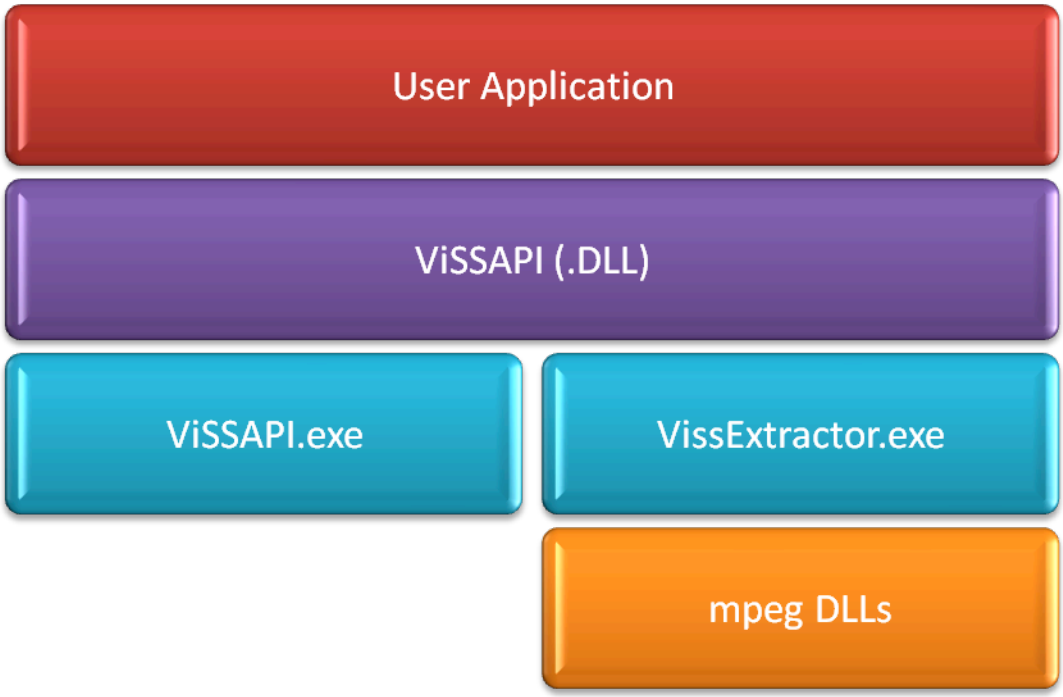

Index & Extract parameters
Extract
Index
Search parameters
Search
Search results

5.2 ViSSAPI Structure

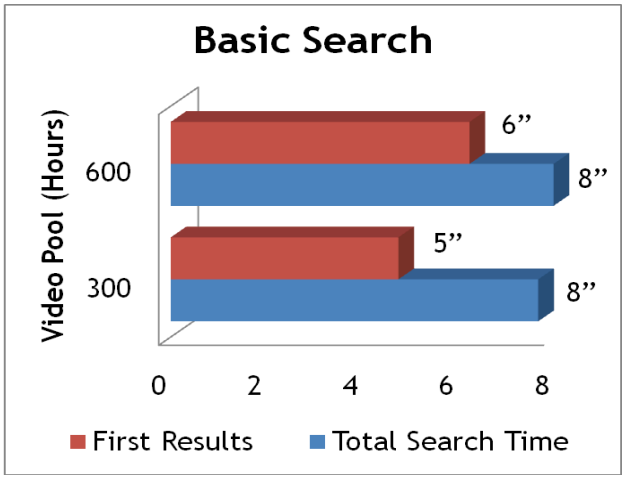
According to the functionality described into the previous paragraph, the ViSSAPI structure is as depicted in the figure below. The ViSSAPI functionality is contained in two main blocks:

- the VissExtractor.exe, that extracts the fingerprint from the video file
- the ViSSAPI.exe, that inserts the video index into the database and implements the search logic

Part of the functionality of VissExtractor.exe is based on a set of ffmpeg DLLs. These are part of the ViSSAPI distribution file set, as presented in the ViSSAPI User Guide.



5.3 ViSSAPI Performance



ViSSAPI provides the fastest method in the industry to search a video clip in a pool of videos.

The adjacent figure presents the benchmarking results for the search times of a few seconds clip (~10 seconds duration) into a pool of 300 hours and 600 hours video repository.

More specifically, for a 300 hours repository, the search algorithm produced the first occurrence result in just 5 seconds, while completing the total search process in a total time of just 8 seconds.

For the case of 600 hours of video repository, the first search result was produced in just 6 seconds and the entire search has been completed in a total of just 8 seconds!

6. ViSSAPI Function Reference

6.1 ViSSDBConnect

Connects to a ViSS videos database

```
int ViSSDBConnect(
      char  *dbName,
      char  *UserName,
      char  *Password
);
```

Parameters

dbName

A pointer of type char that identifies the database name.

UserName

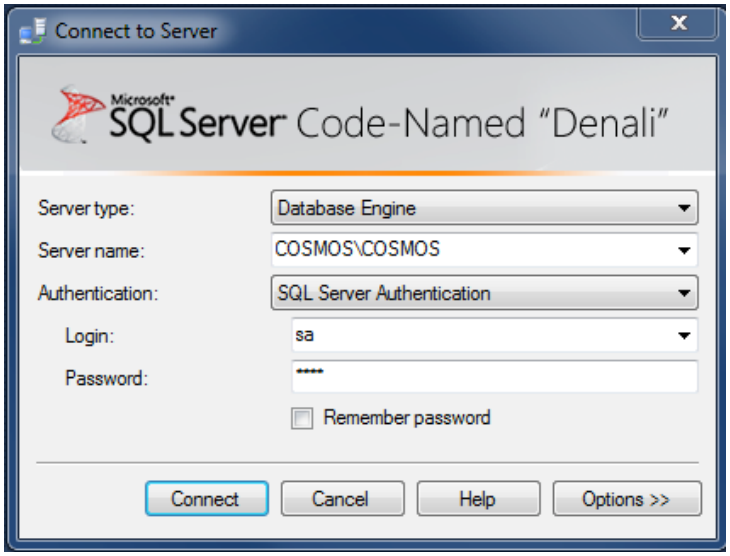
A pointer of type char that identifies the user name to access the database.

Password

A pointer of type char that identifies the password to access the database.

The parameters of this function are the same as used by any user accessing a MSSQL database. For example and according to the next figure, the values are as follows:

```
dbName = "COSMOS\COSMOS"
UserName = "sa"
Password = "viss"
```



Example

A calling example for this function is:

```
int cmdReturn = ViSSDBConnect("COSMOS\\COSMOS", "sa", "viss");
```

Return Values

Value	Description
ACTION_VISSDBCONNECT	The operation was completed successfully.
STATUS_ViSSDBCONNECT _FAIL	The operation was not completed successfully.

Remarks
This is the first function to be executed in order to connect to the video indexing database. Without successful completion of this function, no other function will operate.

See also
ViSSDBDisConnect

6.2 ViSSDBDisConnect

Disconnects from a ViSS videos database

```
int ViSSDBConnect(
      char  *dbName,
      char  *UserName,
      char  *Password
);
```

Parameters

dbName

A pointer of type char that identifies the database name.

UserName

A pointer of type char that identifies the user name to access the database.

Password

A pointer of type char that identifies the password to access the database.

Example

A calling example for this function is:

```
int cmdReturn = ViSSDBDisConnect("COSMOS\\COSMOS", "sa", "viss");
```

Return Values

Value	Description
ACTION_VISSDBDISCONNECT	The operation was completed successfully.
STATUS_ViSSDBDISCONNECT_FAIL	The operation was not completed successfully.

Remarks

The calling parameters of this function are exactly the same as of ViSSDBConnect. This function should be called at the end of the application session in order to close an established database connection.

See also

ViSSDBConnect

6.3 ViSSVideoList

Returns the list of videos already indexed into the database.

```
int ViSSVideoList(
      char *VideoListOutput
);
```

Parameters

VideoListOutput

A pointer of type char that identifies the output list filename. If NULL, the output list appears only on the monitor.

Example

A calling example for this function is:

```
int cmdReturn = ViSSVideoList("MyVideoList.txt");
```

Return Values

Value	Description
ACTION_VISSVIDEOLIST	The operation was completed successfully.

Remarks

The output of this function is the list of the videos in the database. For each video, the videoID and the video name are listed. **This videoID should be used by ViSSISearch as a parameter in order to specify the video to be used as a query. The ViSSVideoList function is thus used for obtaining the desired videoID, before executing the ViSSISearch.**

See Also

ViSSIndex, ViSSISearch, ViSSDeleteVideo

6.4 ViSSIndex

Indexes and inserts a video file into the database.

```
int ViSSIndex(
      char  *VideoFileName
);
```

Parameters

VideoFileName

A pointer to a variable of type char that contains the video file name.

Example

A calling example for this function is:

```
int cmdReturn = ViSSIndex("can_he_fly.avi");
```

Return Values

Value	Description
ACTION_VISSINDEX	The operation was completed successfully.
ACTION_VISSINDEX_INVALID_FILE	The specified `VideoFileName` is invalid.
ACTION_VISSINDEX_SETUP_FK_FAIL	The indexing preprocessing creating a “foreign key” parameter, has failed.
ACTION_VISSINDEX_XTRACTOR_EV_FAIL	The indexing preprocessing extraction producing the “events_vertical.txt” temporary information file, has failed.

Remarks

This function implements the indexing algorithm of ViSS and is mandatory in order to insert the index of a new video into the database. Searching is based on the basis of this index.

See Also

ViSSVideoList, ViSSSearch, ViSSDeleteVideo

6.5 ViSSSearch

Searches a video file in the database on the basis of video indices. Videos matching the query are returned in the form of a list.

```
int ViSSISearch (
      int   videoID,
      char   *ResultsOutput
);
```

Parameters

videoID

An integer that identifies the video ID. This video ID field must be taken from those listed through the ViSSVideoList function.

ResultsOutput

A pointer of type char that identifies the filename storing the search results.

Example

A calling example for this function is:

```
int cmdReturn = ViSSSearch(1, "MySearchResults.txt");
```

Return Values

Value	Description
STATUS_VISSSEARCH_SUCCESS	The operation was completed successfully.
STATUS_VISSSEARCH_FAIL	The operation completed with no results.

Remarks

The output of this function is a list with the videoID of all matching videos in the database.

See Also

ViSSVideoList, ViSSIndex, ViSSDeleteVideo, ViSSExtSearch

6.6 ViSSExtSearch

Searches a video file in the database on the basis of video indices and with 'Extended' parameters and options. Videos matching the query are returned in the form of a list.

```
int ViSSExtSearch (
      int   videoID,
      int   search_mode,
      int   qstart_qf,
      int   qstop_qf,
      int   kmax,
      int   md,
      int   fa,
      int   jit,
      int   colorTol,
      char   *ResultsOutput
);
```

Parameters

videoID

An integer that identifies the video ID. This video ID field must be taken from those listed through the ViSSVideoList function.

search_mode

An integer that identifies the search mode as follows:

1:	search based on scene duration - is default
3:	search based on mean value of colours
13:	search_mode for 1 & 3 combined (logic AND)
173:	search based on movement content combined with colour (logic AND) (advisable for very short search videos of 2-3 scenes)
0:	search system will decide on above based on length and movement content of the search video

The user might not want to search the whole video clip. In this case he gives with

qstart_qf

the frame number of the search video clip which will be considered as the start frame for the search.

qstop_qf

the frame number of the search video clip which will be considered as the end frame for the search.

Irrespective of the *qstop_qf* setting the search will occur on the basis of scene changes occurring from frame numbers *qstart_qf* up to maximally *qstart_qf* + *kmax*. So

kmax

is the depth (in scene changes) of the search to be performed. Its maximal value is 25 scene changes.

md & fa

parameters for tolerating miss interpretations of scene changes during search.
Values *md = 0, fa = 0* allow no tolerance.
Maximal values are *md = 4, fa = 4*.
Default values are *md = 1, fa = 1*.

jit

a 'jitter' tolerance in frame numbers (0, 1, 2, or maximally 3) for the placement of scene changes. Default is *jit = 0*.

colorTol

a color tolerance for comparing the color mean values of scene changes.
Default is *colorTol = 3*.

ResultsOutput

A pointer of type char that identifies the filename storing the search results.

Example

A calling example for this function is:

```
int cmdReturn = ViSSSearch(1,
                           13,
                           23,
                           4765,
                           12,
                           1,
                           1,
                           0,
                           5,
                           "MySearchResults.txt");
```

Return Values

Value	Description
STATUS_VISSEXTSEARCH_SUCCESS	The operation was completed successfully.
STATUS_VISSEXTSEARCH_FAIL	The operation completed with no results.

Remarks

The output of this function is a list with the videoID of all matching videos in the database. For each match the exact correspondence between frame numbers of query and matched videos is appended along with further matching parameters.

See Also

ViSSVideoList, ViSSIndex, ViSSDeleteVideo, ViSSSearch

6.7 ViSSDeleteVideo

Deletes a video file from the database.

```
int ViSSDeleteVideo(
      char *VideoFileName
);
```

Parameters

VideoFileName

A pointer to a variable of type char that contains video filename.

Example

A calling example for this function is:

```
int cmdReturn = ViSSDeleteVideo("MyVideo.avi");
```

Return Values

Value	Description
ACTION_VISSDELETEVIDEO	The operation was completed successfully.
ACTION_VISSDEL_VID_NO_RESULTS	The operation was completed without any result. The `VideoFileName` is not in the database.

Remarks -

See Also

ViSSVideoList, ViSSIndex, ViSSSearch, ViSSDeleteVideo

7. ViSSAPI User Guide

This guide provides instruction for the C/C++ developer, in order to create custom applications using the ViSSAPI.

7.1 ViSSAPI distribution files

The ViSSAPI is provided as a set of files distributed to the developer. Those files are as follows:

File name	Description
ViSSAPI.dll	ViSSAPI functions as Dynamic Link Library (DLL)
ViSSAPI.lib	ViSSAPI functions Library linked with the user application
ViSSAPI.h	ViSSAPI C/C++ header file
VissExtractor.exe	ViSSAPI Indexing logic
ViSSAPI.exe	ViSSAPI Executable that is handled by the DLL
avcodec-52.dll, avcore-0.dll, avdevice-52.dll, avfilter-1.dll, avformat-52.dll, avutil-50.dll, swscale-0.dll, swscale.dll	Dynamic Link Libraries (DLL) from http://ffmpeg.org/ free software project

Important Notes:

1) All above files should be in the same directory with the executable file created by the developer.

2) Before executing any of the ViSSAPI function, the developer must start the database instance.

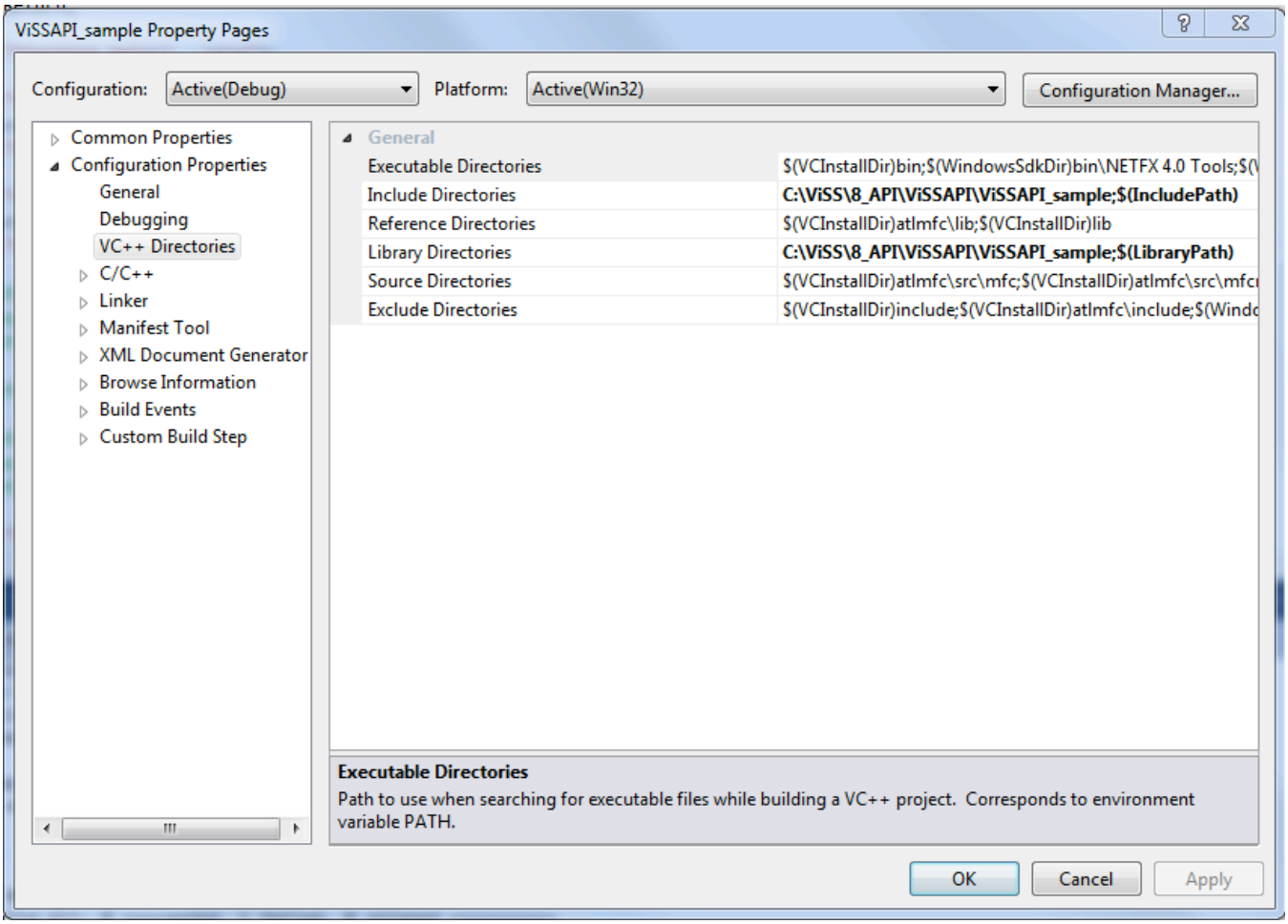
7.2 How to create custom C/C++ application using ViSSAPI

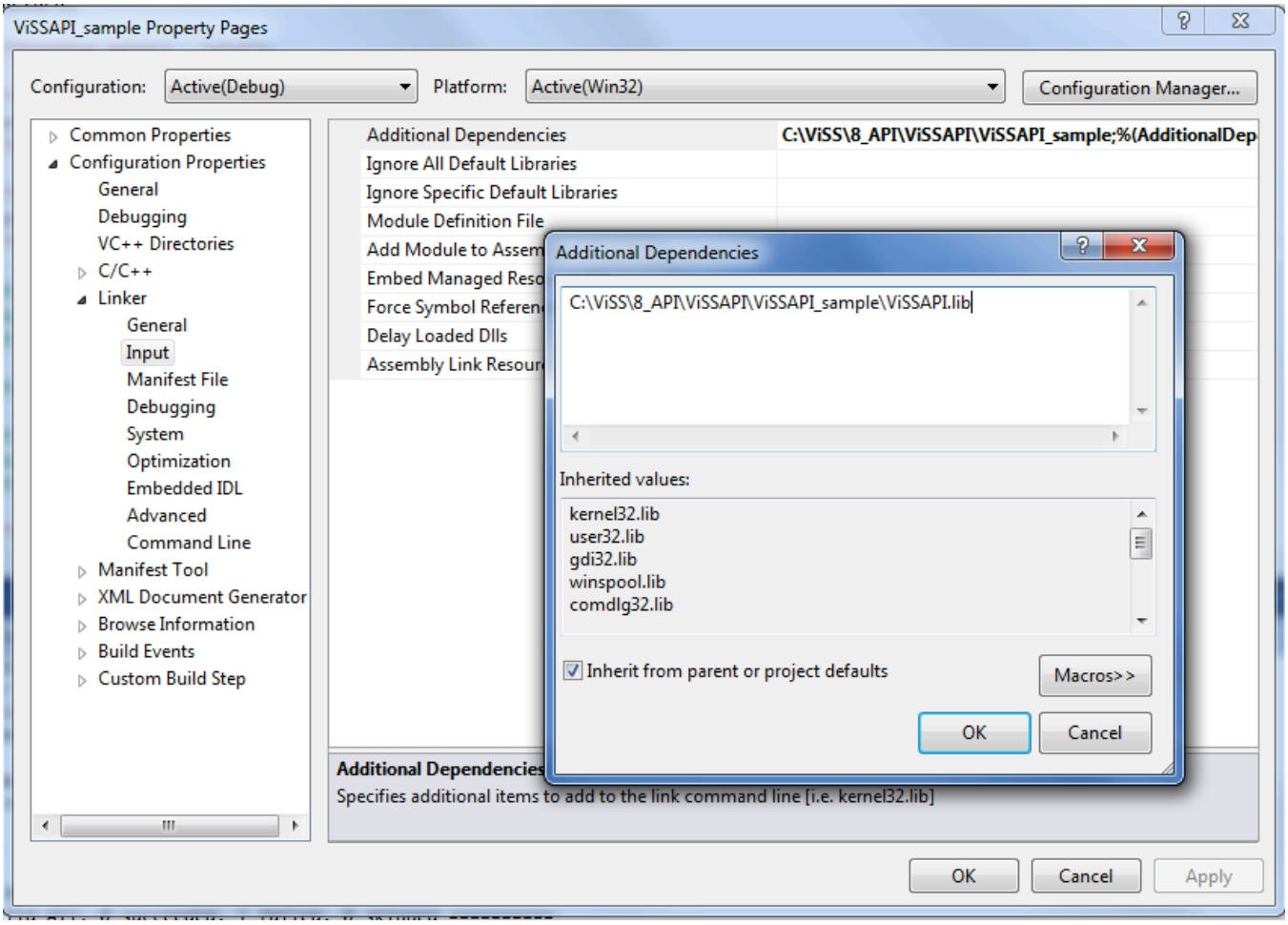
The steps to create a custom C/C++ application in Visual Studio 10, using ViSSAPI are the following:

Step 1: Copy the provided ViSSAPI files to a directory.

Step 2: Create a Microsoft Visual Studio C/C++ application specifying, in the C/C++ category, the path location of the ViSSAPI files, to both the Include Directories and Library Directories settings. Moreover specify in the Linker, Input category the Additional Dependencies.

Refer to the following images with the details.





Step 3: Once creating your program insert the inline code
`#include <ViSSAPI.h>`
so the compiler has a reference of the ViSSAPI functions and definitions.

7.3 ViSSAPI.h

```
#ifndef __DLL__H_
#define __DLL__H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define ACTION_VISSDBCONNECT                    1
#define ACTION_VISSDBDISCONNECT                 2
#define ACTION_VISSINDEX_SETUP                  3
#define ACTION_VISSINDEX                        4
#define ACTION_VISSSEARCH                       5
#define ACTION_VISSEXTSEARCH                    51
#define ACTION_VISSVIDEOLIST                    6
#define ACTION_VISSDELETEVIDEO                  7

#define STATUS_ViSSDBCONNECT_FAIL               0
#define STATUS_ViSSDBDISCONNECT_FAIL            0

#define ACTION_VISSINDEX_INVALID_FILE           -1      // The specified "VideoFileName" is invalid
#define ACTION_VISSINDEX_SETUP_FK_FAIL          -2      // The indexing preprocessing creating a “foreign key”
                                                        // parameter, had FAIL
#define ACTION_VISSINDEX_XTRACTOR_EV_FAIL       -3      // The indexing preprocessing extraction producing the
                                                        // “events_vertical.txt” temporary information file,
                                                        // had FAIL

#define ACTION_VISSINDEX_NO_RESULTS             -4      // The operation completed with no results
#define ACTION_VISSINDEX_INVALID_VIDEO_ID       -6      // The operation completed with no result because the
                                                        // videoID query parameter has not been found

#define ACTION_VISSDEL_VID_NO_RESULTS           -5      // The operation completed with no results. The
                                                        // VideoFileName is not in the database


#define STATUS_VISSINDEX_SUCCESS                1
#define STATUS_VISSSEARCH_SUCCESS               1
#define STATUS_VISSSEARCH_FAIL                  -1
#define STATUS_VISSEXTSEARCH_SUCCESS            1
#define STATUS_VISSEXTSEARCH_FAIL               -1


extern "C" __declspec(dllexport) int ViSSDBConnect(char *dbName, char *UserName, char *Password);
extern "C" __declspec(dllexport) int ViSSDBDisConnect(char *dbName, char *UserName, char *Password);

extern "C" __declspec(dllexport) int ViSSVideoList(char *VideoListOutput);

extern "C" __declspec(dllexport) int ViSSIndex(char *VideoFileName);
extern "C" __declspec(dllexport) int ViSSSearch(int videoID, char *ResultsOutput);
extern "C" __declspec(dllexport) int ViSSExtSearch (     int videoID,
                                                         int search_mode,
                                                         int qstart_qf,
                                                         int qstop_qf,
                                                         int kmax,
                                                         int md,
                                                         int fa,
                                                         int jit,
                                                         int colorTol,
                                                         char *ResultsOutput);

extern "C" __declspec(dllexport) int ViSSDeleteVideo(char *VideoFileName);

#endif
```

7.4 ViSSAPI Sample file

```
#include <stdio.h>
#include <string.h>
#include <ViSSAPI.h>

void main() {
	int cmdReturn;

	printf("Starting main()...\n\n");

	// Connecting to database. Be carefull of the DB name as parameter!
	printf("Starting ViSSDBConnect...\n");
	cmdReturn = ViSSDBConnect("COSMOS\\COSMOS", "sa", "viss");
	if (cmdReturn == ACTION_VISSDBCONNECT)
		printf("Completed ViSSDBConnect!\n");
	else
		printf("ViSSDBConnect FAIL!\n");

goto just_search;

	// List all Videos
	printf("Starting ViSSVideoList...\n");
	cmdReturn = ViSSVideoList("MyVideoList.txt");
	if (cmdReturn == ACTION_VISSVIDEOLIST)
		printf("Completed ViSSVideoList!\n");
	else
		printf("ViSSVideoList FAIL!\n");

	// Insert (=index) the new video.
	printf("Starting ViSSIndex...\n");
	cmdReturn = ViSSIndex("Amalfi_360_flv.flv");
	if (cmdReturn == ACTION_VISSINDEX)
		printf("\nCompleted ViSSIndex!\n");
	else
		printf("\nViSSIndex FAIL!\n");

just_search:

	// List all Videos again, after previous insertion (=indexing)
	printf("Starting ViSSVideoList...\n");
	cmdReturn = ViSSVideoList("MyVideoList.txt");
	if (cmdReturn == ACTION_VISSVIDEOLIST)
		printf("Completed ViSSVideoList!\n");
	else
		printf("ViSSVideoList FAIL!\n");

	// Search of the video with its video_id, as listed in the previous List action
	printf("Starting ViSSSearch...\n");
	cmdReturn = ViSSSearch(54, "SearchResults_Default_video_id_54.txt");
	if (cmdReturn == ACTION_VISSSEARCH)
		printf("Completed ViSSSearch!\n");
	else
	if (cmdReturn == ACTION_VISSINDEX_INVALID_VIDEO_ID)
		printf("ViSSSearch INVALID VIDEO ID!\n");

	// Search of the video with its video_id, as listed in the previous List action
	printf("Starting ViSSSearch...\n");
	cmdReturn = ViSSExtSearch(54, 13, 23, 4765, 12, 1, 1, 0, 5, "SearchResults_EXtended_video_id_54.txt");
	if (cmdReturn == ACTION_VISSSEARCH)
		printf("Completed ViSSSearch!\n");
	else
	if (cmdReturn == ACTION_VISSINDEX_INVALID_VIDEO_ID)
		printf("ViSSSearch INVALID VIDEO ID!\n");
```

```
        // Delete a video
        printf("Starting ViSSDeleteVideo...\n");
        cmdReturn = ViSSDeleteVideo("news");
        if (cmdReturn == ACTION_VISSDELETEVIDEO)
                printf("Completed ViSSDeleteVideo!\n");
        else
                printf("ViSSDeleteVideo FAIL!\n");

        // List all Videos again, after previous Deletion
        printf("Starting ViSSVideoList...\n");
        cmdReturn = ViSSVideoList("MyVideoList.txt");
        if (cmdReturn == ACTION_VISSVIDEOLIST)
                printf("Completed ViSSVideoList!\n");
        else
                printf("ViSSVideoList FAIL!\n");

        // Disconnecting to database. Be carefull of the DB name as parameter!
        printf("Starting ViSSDBDisConnect...\n");
        cmdReturn = ViSSDBDisConnect("COSMOS\\COSMOS", "sa", "viss");
        if (cmdReturn == ACTION_VISSDBDISCONNECT)
                printf("Completed ViSSDBDisConnect!\n");
        else
                printf("ViSSDBDisConnect FAIL!\n");

        printf("Ending main()!\n");

} // main
```